Lecture 9

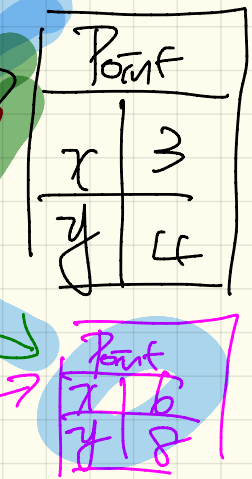
Thursday Oct. 5

```
class Util {
  void reassignRef(Point q) {
    Point np = new Point(6, 8);
    q = np;
  }
}
```

expecting an address of some Point object

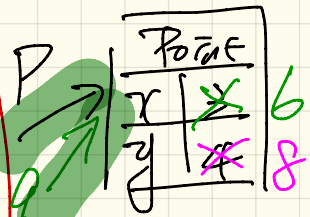
a copy of p will replace every occurrence of q

```
testCallByRef_1(){
  Util u = new Util();
  Point p = new Point(3, 4);
  u.reassignRef(p);
}
```



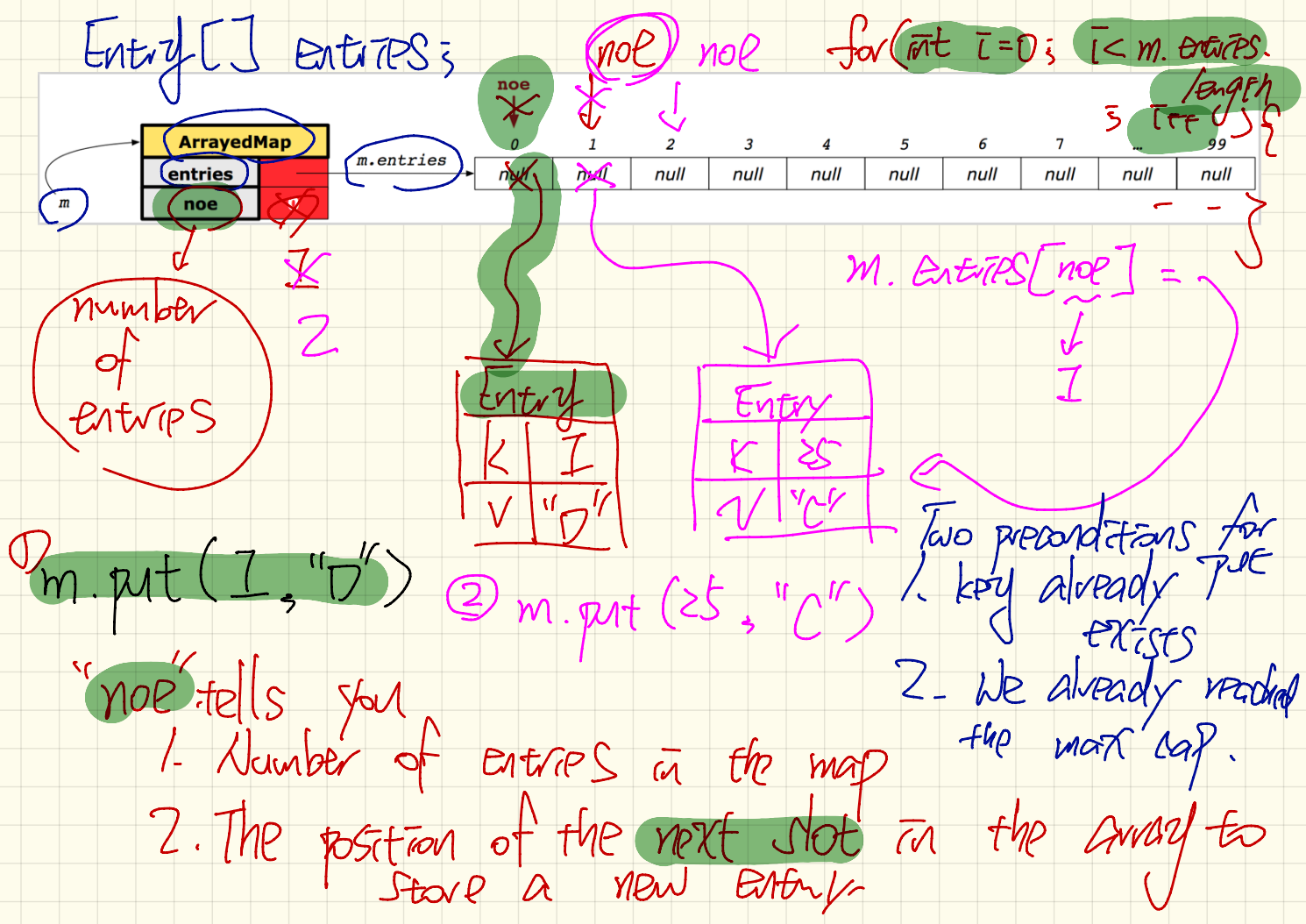
```
class Ut {
  void changeViaRef(Point q) {
    q.moveH(3);
    q.moveV(4);
  }
}
```

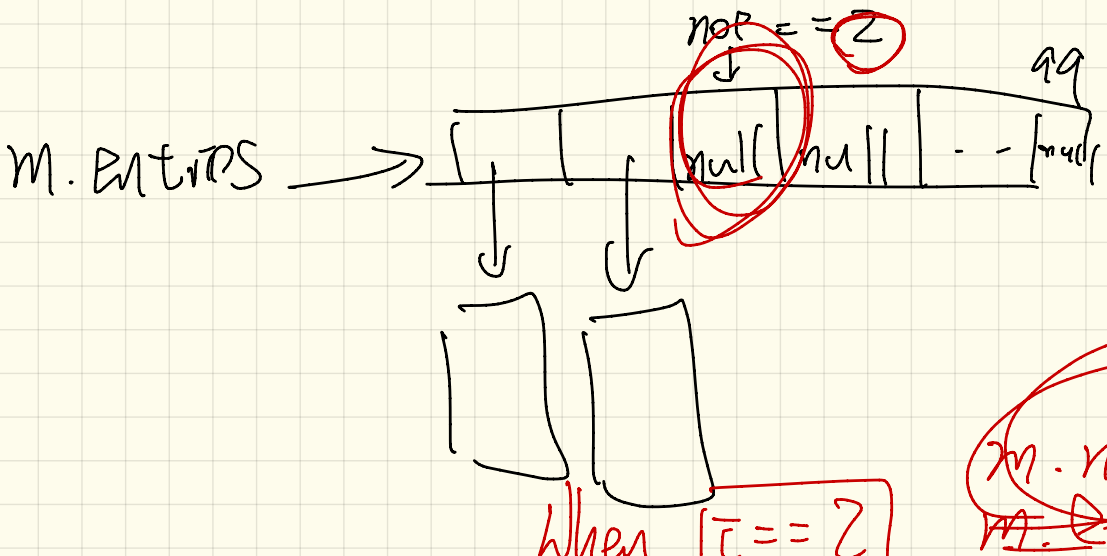
q is replaced by a copy of p.




```
testCallByRef_2() {
  Point p = new Point(3,4);
  u.changeViaRef(p);
}
```

p.x == 6 &&
q.y == 8.
assertTrue(
p.x == q.x &&
p.y == q.y &&


Entry[] entries;
noe
noe
for(int i=0; i< m.entries.length; i++){
ArrayedMap
entries
noe
m
m.entries
0 1 2 3 4 5 6 7 ... 99
null null null null null null null null null null
number of entries
2
Entry
K I
V "D"
Entry
K 25
V "C"
m.entries[noe] =
noe
1
① m.put(1, "D")
② m.put(25, "C")
"noe" tells you
1. Number of entries in the map
2. The position of the next slot in the array to store a new entry.
Two preconditions for put
1. key already put exists
2. We already reached the max cap.




```
for( int i=0; i < m.entries.length ; i++) {
    println( m.entries[i].getKey() );
}
```

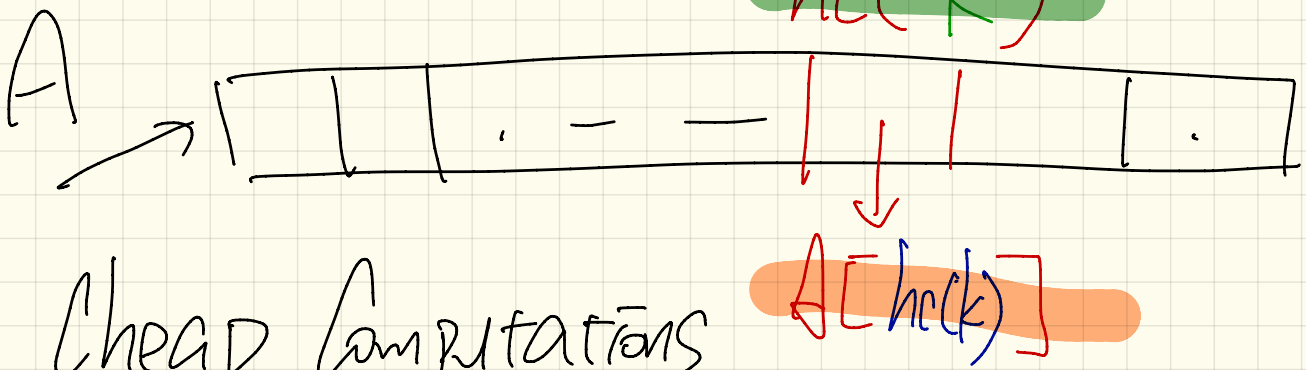
null.

function map vs. relation

(1, "D") (7, "D")

Hashing

K → hashing → hc(k)



Cheap Computations

1. Arithmetic (K % 11)
2. Array indexing (A[i])